**Batch: D - 1 Roll No.: 16010122096**

**Experiment No. 05**

|  |
| --- |
| **TITLE**: Write a program to demonstrate the LINE CLIPPING algorithm |

**AIM:**

**Visit Vlab and Explore it**

[**https://cse18-iiith.vlabs.ac.in/exp/clipping-line/**](https://cse18-iiith.vlabs.ac.in/exp/clipping-line/)

Write a program to demonstrate the LINE CLIPPING algorithm

1. Cohen-Sutherland-algorithm
2. Mid-Point Subdivision Line Clipping Algorithm
3. Liang-Barsky Line Clipping Algorithm

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

*CO1: Understand the basic concepts of computer graphics and OpenGL*

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

**<https://www.javatpoint.com/computer-graphics-line-clipping>**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Algorithm 1, 2 and 3**

**A] COHEN – SUTHERLAND – ALGORITHM**

**Step1:** Calculate positions of both endpoints of the line

**Step2:** Perform OR operation on both of these end-points

**Step3:** If the OR operation gives 0000  
       Then  
                line is considered to be visible  
       else  
          Perform AND operation on both endpoints  
      If And ≠ 0000  
          then the line is invisible  
        else  
      And=0000  
    Line is considered the clipped case.

**Step4:** If a line is clipped case, find an intersection with boundaries of the window  
                m = (y2-y1) \* (x2-x1)

(a) If bit 1 is "1" line intersects with left boundary of rectangle window  
                y3=y1+m(x-X1)  
                where X = Xwmin  
                where Xwminis the minimum value of X co-ordinate of window

(b) If bit 2 is "1" line intersect with right boundary  
                y3=y1+m(X-X1)  
                where X = Xwmax  
                where X more is maximum value of X co-ordinate of the window

(c) If bit 3 is "1" line intersects with bottom boundary  
                X3=X1+(y-y1)/m  
                      where y = ywmin  
                ywmin is the minimum value of Y co-ordinate of the window

(d) If bit 4 is "1" line intersects with the top boundary  
                X3=X1+(y-y1)/m  
                      where y = ywmax  
                ywmax is the maximum value of Y co-ordinate of the window

**B] MID-POINT SUBDIVISION LINE CLIPPING ALGORITHM**

**Step1:** Calculate the position of both endpoints of the line

**Step2:** Perform OR operation on both of these endpoints

**Step3:** If the OR operation gives 0000  
            then  
                    Line is guaranteed to be visible  
          else  
                  Perform AND operation on both endpoints.  
                  If AND ≠ 0000  
            then the line is invisible  
      else  
            AND=6000  
            then the line is clipped case.

**Step4:** For the line to be clipped. Find midpoint  
            Xm=(x1+x2)/2  
            Ym=(y1+y2)/2  
        Xmis midpoint of X coordinate.  
                  Ymis midpoint of Y coordinate.

**Step5:** Check each midpoint, whether it nearest to the boundary of a window or not.

**Step6:** If the line is totally visible or totally rejected not found then repeat step 1 to 5.

**Step7:** Stop algorithm.

**C] LIANG-BARSKY LINE CLIPPING ALGORITHM**

**Step1:** **Initialize the parameters**:

* Define the clipping window with boundaries xmin, xmax, ymin, and ymax.
* Set the endpoints of the line segment as (x1, y1) and (x2, y2).

**Step2:** **Calculate the differences**:

* Compute the differences: dx = x2 - x1 and dy = y2 - y1.

**Step3:** **Set initial values for the parametric variable t**:

* t1 = 0.0 (start of the line segment)
* t2 = 1.0 (end of the line segment)

**Step4:** **Check the boundary conditions**:

* Check if the line crosses the clipping window boundaries by testing against the four sides (left, right, bottom, top).
* For each boundary, calculate the parameter p and q:
  + p = -dx, q = x1 - xmin (Left boundary)
  + p = dx, q = xmax - x1 (Right boundary)
  + p = -dy, q = y1 - ymin (Bottom boundary)
  + p = dy, q = ymax - y1 (Top boundary)

**Step5:** **Clip the line segment**:

* For each boundary:
  + If p = 0 and q < 0, the line is outside and can be discarded.
  + If p < 0, compute r = q / p. If r > t2, the line is outside the clipping window. If r > t1, set t1 = r.
  + If p > 0, compute r = q / p. If r < t1, the line is outside the clipping window. If r < t2, set t2 = r.

**Step6:** **Determine if the line segment is inside the clipping window**:

* If t1 <= t2, the line segment is at least partially within the clipping window.
* Compute the new clipped line segment endpoints:
  + x1' = x1 + t1 \* dx
  + y1' = y1 + t1 \* dy
  + x2' = x1 + t2 \* dx
  + y2' = y1 + t2 \* dy

**Step7:** **Draw the clipped line**:

* The new endpoints (x1', y1') and (x2', y2') represent the clipped line segment within the clipping window.

**Implementation details:**

**A**

#include <GL/glut.h>

#include <iostream>

const int LEFT = 1, RIGHT = 2, BOTTOM = 4, TOP = 8;

float xmin = -100, ymin = -100, xmax = 100, ymax = 100;

int computeCode(float x, float y) {

int code = 0;

if (x < xmin) code |= LEFT;

else if (x > xmax) code |= RIGHT;

if (y < ymin) code |= BOTTOM;

else if (y > ymax) code |= TOP;

return code;

}

void cohenSutherlandClip(float x1, float y1, float x2, float y2) {

int code1 = computeCode(x1, y1);

int code2 = computeCode(x2, y2);

bool accept = false;

while (true) {

if (!(code1 | code2)) {

accept = true;

break;

} else if (code1 & code2) {

break;

} else {

int code\_out;

float x, y;

if (code1 != 0)

code\_out = code1;

else

code\_out = code2;

if (code\_out & TOP) {

x = x1 + (x2 - x1) \* (ymax - y1) / (y2 - y1);

y = ymax;

} else if (code\_out & BOTTOM) {

x = x1 + (x2 - x1) \* (ymin - y1) / (y2 - y1);

y = ymin;

} else if (code\_out & RIGHT) {

y = y1 + (y2 - y1) \* (xmax - x1) / (x2 - x1);

x = xmax;

} else if (code\_out & LEFT) {

y = y1 + (y2 - y1) \* (xmin - x1) / (x2 - x1);

x = xmin;

}

if (code\_out == code1) {

x1 = x;

y1 = y;

code1 = computeCode(x1, y1);

} else {

x2 = x;

y2 = y;

code2 = computeCode(x2, y2);

}

}

}

if (accept) {

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2f(x1, y1);

glVertex2f(x2, y2);

glEnd();

}

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_LINES);

glVertex2f(-150, -150);

glVertex2f(150, 150);

glEnd();

glColor3f(0.0, 1.0, 0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(xmin, ymin);

glVertex2f(xmax, ymin);

glVertex2f(xmax, ymax);

glVertex2f(xmin, ymax);

glEnd();

cohenSutherlandClip(-150, -150, 150, 150);

glFlush();

}

void init() {

glClearColor(1.0, 1.0, 1.0, 1.0);

glColor3f(1.0, 0.0, 0.0);

gluOrtho2D(-200, 200, -200, 200);

}

int main(int argc, char \*\*argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("Cohen-Sutherland Line Clipping");

init();

glutDisplayFunc(display);

glutMainLoop();

return 0;

}

**B**

#include <GL/glut.h>

float xmin = -100, ymin = -100, xmax = 100, ymax = 100;

bool inside(float x, float y) {

return x >= xmin && x <= xmax && y >= ymin && y <= ymax;

}

void midPointClip(float x1, float y1, float x2, float y2) {

if (inside(x1, y1) && inside(x2, y2)) {

glBegin(GL\_LINES);

glVertex2f(x1, y1);

glVertex2f(x2, y2);

glEnd();

} else {

float xm = (x1 + x2) / 2;

float ym = (y1 + y2) / 2;

if ((abs(x1 - x2) < 1 && abs(y1 - y2) < 1) || inside(xm, ym)) {

return;

}

midPointClip(x1, y1, xm, ym);

midPointClip(xm, ym, x2, y2);

}

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_LINES);

glVertex2f(-150, -150);

glVertex2f(150, 150);

glEnd();

glColor3f(0.0, 1.0, 0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(xmin, ymin);

glVertex2f(xmax, ymin);

glVertex2f(xmax, ymax);

glVertex2f(xmin, ymax);

glEnd();

midPointClip(-150, -150, 150, 150);

glFlush();

}

void init() {

glClearColor(1.0, 1.0, 1.0, 1.0);

glColor3f(1.0, 0.0, 0.0);

gluOrtho2D(-200, 200, -200, 200);

}

int main(int argc, char \*\*argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("Mid-Point Subdivision Line Clipping");

init();

glutDisplayFunc(display);

glutMainLoop();

return 0;

}

**C**

#include <GL/glut.h>

#include <iostream>

float xmin = -100, ymin = -100, xmax = 100, ymax = 100;

bool clipTest(float p, float q, float &t1, float &t2) {

float r;

if (p < 0.0) {

r = q / p;

if (r > t2) return false;

else if (r > t1) t1 = r;

} else if (p > 0.0) {

r = q / p;

if (r < t1) return false;

else if (r < t2) t2 = r;

} else if (q < 0.0) return false;

return true;

}

void liangBarskyClip(float x1, float y1, float x2, float y2) {

float dx = x2 - x1, dy = y2 - y1, t1 = 0.0, t2 = 1.0;

if (clipTest(-dx, x1 - xmin, t1, t2))

if (clipTest(dx, xmax - x1, t1, t2))

if (clipTest(-dy, y1 - ymin, t1, t2))

if (clipTest(dy, ymax - y1, t1, t2)) {

if (t2 < 1.0) {

x2 = x1 + t2 \* dx;

y2 = y1 + t2 \* dy;

}

if (t1 > 0.0) {

x1 = x1 + t1 \* dx;

y1 = y1 + t1 \* dy;

}

glBegin(GL\_LINES);

glVertex2f(x1, y1);

glVertex2f(x2, y2);

glEnd();

}

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_LINES);

glVertex2f(-150, -150);

glVertex2f(150, 150);

glEnd();

glColor3f(0.0, 1.0, 0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(xmin, ymin);

glVertex2f(xmax, ymin);

glVertex2f(xmax, ymax);

glVertex2f(xmin, ymax);

glEnd();

liangBarskyClip(-150, -150, 150, 150);

glFlush();

}

void init() {

glClearColor(1.0, 1.0, 1.0, 1.0);

glColor3f(1.0, 0.0, 0.0);

gluOrtho2D(-200, 200, -200, 200);

}

int main(int argc, char \*\*argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("Liang-Barsky Line Clipping");

init();

glutDisplayFunc(display);

glutMainLoop();

return 0;

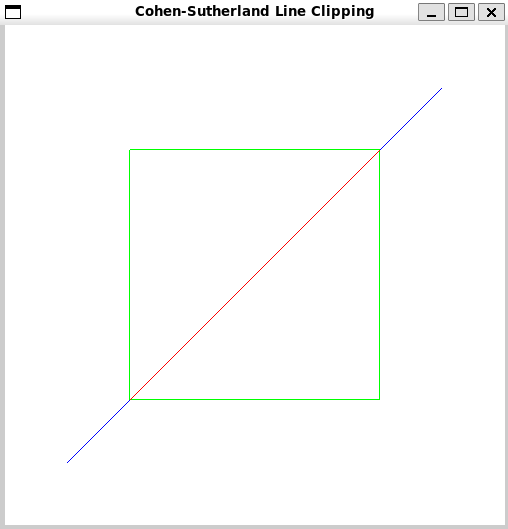
}

**Output(s) (final edited screen shot):**

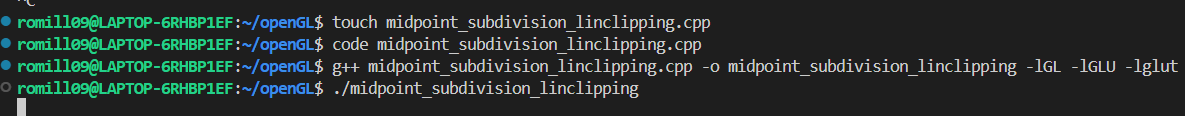
**A]**

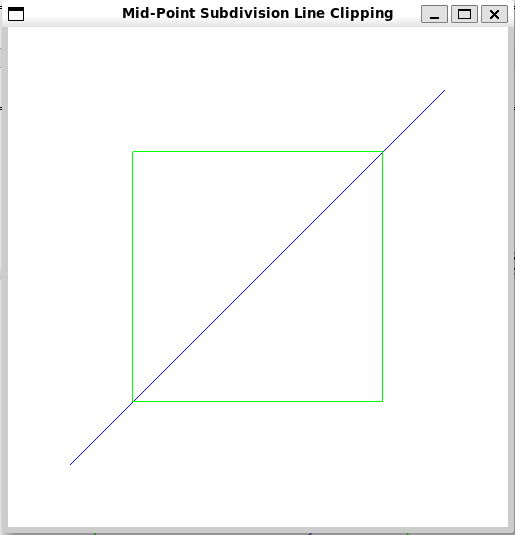
****

****



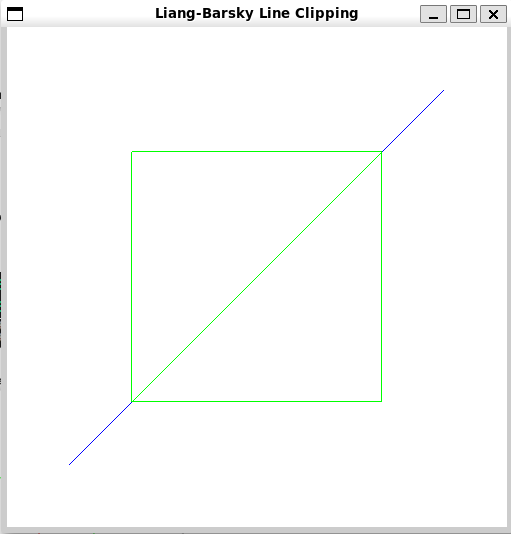
**B]**

****

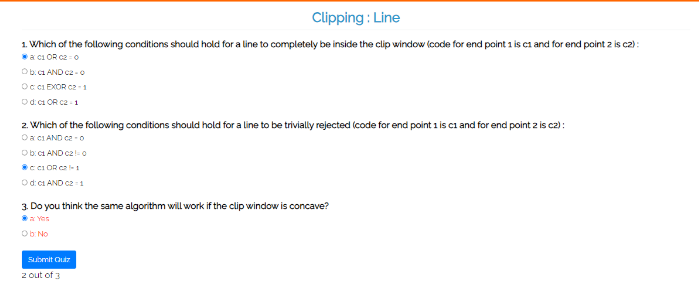
****

**C]**

****

****

**Screenshots from VLab(if any):**

****

**Conclusion and discussion (Comparative ):**

* **Cohen-Sutherland Algorithm:**
* Uses region codes to classify endpoints.
* Straightforward but may involve multiple intersection checks.
* Potentially slow with multiple clipping operations.
* **Mid-Point Subdivision Algorithm:**
* Ensures precision by recursively dividing the line segment.
* Computation-intensive and slow due to repeated subdivision.
* Less suitable for real-time applications.
* **Liang-Barsky Algorithm:**
* Directly computes intersection points using parametric equations.
* Most efficient with fewer comparisons.
* Faster than Cohen-Sutherland and midpoint subdivision.
* Ideal for real-time rendering where speed and accuracy are crucial.

**Date: 25 / 08 / 2024 Signature of faculty in-charge**

**Post Lab**

**What is Turtle in CG, demonstrate use of Turtle by implementing it?**

In computer graphics, **Turtle graphics** is a popular method used for drawing shapes and patterns, primarily used in educational contexts to teach programming concepts. The Turtle graphics system involves a "turtle" that moves around the screen, drawing lines as it goes. It follows commands to move forward, turn, and draw, making it a great tool for visualizing programming concepts.

**Key Concepts:**

* **Turtle**: A pen-like object that moves around the screen.
* **Commands**: Instructions for moving and drawing, such as forward(distance), right(angle), and left(angle).

**Implementation:**

#include <GL/glut.h>

#include <cmath>

float posX = 200.0f, posY = 270.0f;

float directionAngle = 0.0f;

void drawWhiteLine(float endX, float endY) {

    glBegin(GL\_LINES);

    glVertex2f(posX, posY);

    glVertex2f(endX, endY);

    glEnd();

}

void moveForward(float distance) {

    float newPosX = posX + distance \* cos(directionAngle \* M\_PI / 180.0f);

    float newPosY = posY + distance \* sin(directionAngle \* M\_PI / 180.0f);

    drawWhiteLine(newPosX, newPosY);

    posX = newPosX;

    posY = newPosY;

}

void rotate(float angleChange) {

    directionAngle += angleChange;

}

void drawWhitePattern() {

    glColor3f(1.0f, 1.0f, 1.0f);

    for (int i = 0; i < 360; ++i) {

        glLineWidth(i / 100.0f + 1.0f);

        moveForward(i);

        rotate(59);

    }

}

void display() {

    glClear(GL\_COLOR\_BUFFER\_BIT);

    glLoadIdentity();

    glTranslatef(0.0f, 0.0f, 0.0f);

    drawWhitePattern();

    glutSwapBuffers();

}

void initializeOpenGL() {

    glClearColor(0.0, 0.0, 0.0, 0.0);

    glMatrixMode(GL\_PROJECTION);

    gluOrtho2D(-500, 500, -500, 500);

    glMatrixMode(GL\_MODELVIEW);

}

int main(int argc, char\*\* argv) {

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

    glutInitWindowSize(800, 600);

    glutCreateWindow("Turtle Graphics with White Color");

    initializeOpenGL();

    glutDisplayFunc(display);

    glutMainLoop();

    return 0;

}

**Output:**



